CONTINUOUS TIME MIXED INTEGER PROGRAMMING

Natashia Boland
Georgia Institute of Technology

Joint work with
Mike Hewitt – Loyola University
Luke Marshall & Martin Savelsbergh - Georgia Institute of Technology
Simranjit Kaur & Thomas Kalinowski – University of Newcastle
Motivation

• Timing is everything!

• In logistics and supply chain operations
  ▪ when to make each product
  ▪ when to order new stock
  ▪ when to schedule transport services
  ▪ when to build new facilities or infrastructure

• In mathematical programming, timing decisions are usually modeled by taking a uniform discretization of time, which can lead to
  ▪ enormous models, and/or
  ▪ inaccuracy, resulting in suboptimal solutions

• Is there a better way?
Outline

• Network scheduling
  ▪ Motivated by preventive maintenance scheduling in a bulk goods supply chain
  ▪ Schedule arc shutdowns in a network so as to maximize flow over time
  ▪ Storage at nodes: the impact on time
  ▪ MIP models, lower bounds and upper bounds

• Service network design
  ▪ Scheduling line-haul operations for a less-than-truckload consolidation carrier
  ▪ Fixed charge multicommodity network flow over a time-space network
  ▪ Uniform discretization approaches: the impact on optimality
  ▪ Doing better: dynamic generation of a (much smaller) time-space network
  ▪ Lower bounds, upper bounds and iterative refinement
  ▪ Computational results

• Conclusions
The world’s largest coal export operation

- 35 coal mines
- 11 producers
- 24 load points
- 2 above-rail operators
- 15,000+ train trips per year
- 2 track owner/operators
- Haulage distances up to 350km
- 80 different brands
- 3 coal loading terminals
- Exporting around 140 MTPA, increasing
Trains and dumpstations
Stockyards, stackers and reclaimers
Conveyors, berths and shiploaders
All this equipment requires preventive maintenance
Or else...
Preventive maintenance scheduling

• Planned maintenance of track and terminal equipment:
  ▪ Track sections, e.g. rail grinding, Speno car inspection
  ▪ Stackers and reclaimers
  ▪ Dumpstations and conveyor belts
  ▪ Shiploaders
  ▪ Berths, e.g. rails for shiploaders

• This maintenance causes temporary reductions in the system capacity, and therefore throughput

• Improvements in the system throughput can be realised by appropriately timing maintenance jobs (aligning)
Scheduling maintenance to release capacity

- Capacities (units per day)
  - Max throughput = 12
  - Day 1
  - Max total throughput = 7 + 0 = 7
  - Day 2
  - Max total throughput = 0 + 12 = 12

- Two maintenance jobs:
  - shut down the top inbound arc for one day
  - shut down the bottleneck arc for one day
Scheduling maintenance to release capacity

• Capacities (units per day)

• Max throughput = 12

• Two maintenance jobs:
  ▪ shut down the top inbound arc for one day
  ▪ shut down the bottom inbound arc for one day

• Day 1

• Day 2

• Max total throughput = 0 + 12 = 12

• Max total throughput = 7 + 9 = 16
Scheduling process and rules

-Schedulers from rail and each terminal construct preliminary maintenance schedules in isolation

-These schedules are submitted to central planners, who mediate changes to the schedules in order to better align maintenance jobs

-Schedule changes must satisfy certain rules
  - Jobs can be moved within a ±7 day window about the input initial start time
  - Some jobs must not overlap in time
  - Some jobs must stay within certain periods of the day (e.g. 8am-11am, weekday)
  - Some jobs partially reduce capacity, by a given fraction

-This process is iterated until a consensus is reached

-Objective: maximize the total system throughput over the planning horizon
Typical problem dimensions

• Size of networks:
  ▪ 70-80 nodes
  ▪ 130-140 arcs

• Number of maintenance jobs:
  ▪ 1,000-1,300

• Duration:
  ▪ Range 1 hr – 38 days
  ▪ Average 20-25 hrs

• Job start times on the 15 minutes

• Planning horizon:
  ▪ 1 year

stockyards = storage nodes
Arc shutdown scheduling to maximize flow over time

• Given
  ▪ A capacitated network
    – Arc capacity is the maximum flow rate
    – Source node and sink node
    – Storage nodes with capacities
  ▪ Time horizon $T$
  ▪ Set of arc shutdown jobs
    – Release date $r_a$
    – Deadline $d_a$
    – Processing time $p_a$

• Determine
  ▪ A start time $t^*_a$ for each arc shutdown job $a$
  ▪ A dynamic flow in the network

• So that
  ▪ The rate of flow on any arc at each time
    – Does not exceed the arc capacity
    – Is zero if the time is during the arc shutdown
  ▪ Flow stored at a storage node does not, at any time, exceed its capacity
  ▪ Total flow is maximized

• Key questions
  ▪ Do we have to consider the time continuum? Are there discrete time points we can focus on?
  ▪ How can we get dual bounds?
Arc shutdown scheduling to maximize flow over time

• Given a shutdown schedule, set

\[ \tau = \bigcup_a \{t^*_a, t^*_a + p_a\} = \{t_1, t_2, \ldots, t_n\} \]

with

\[ 0 \leq t_1 \leq t_2 \leq \cdots \leq t_n \leq T \]

• The network is in a constant state in each interval \([t_{i-1}, t_i)\)

• Objective value of the schedule is the max flow in
  • a time-expanded network with
  • a copy for each interval,
  • linked by arcs between storage nodes for consecutive copies
Interesting time points in the case without storage

**Proposition:** In the case without storage, there is an optimal solution in which the start time, \( t^*_a \), of the job on arc \( a \), satisfies \( t^*_a \in S(a) \), where \( S(a) \) is constructed inductively by

\[
S_0(a) := \{ r_a, d_a - p_a \} \\
S_{k+1}(a) := S_k(a) \cup \bigcup_{a' \in J \setminus \{a\}} \left( S_k(a') \cup (S_k(a') + p_{a'}) \cup (S_k(a') - p_a) \cup (S_k(a') + p_{a'} - p_a) \right) \cap [r_a, d_a - p_a] \\
S(a) := S_{|J|-1}(a),
\]

for \( k = 1, \ldots, |J| - 2 \)

where \( J \) is the set of arcs with shutdown jobs to be scheduled.

Hence if the data is integer, so are all the optimal start times.
Example of the case with storage: non-integer optimum

- Node v has storage capacity 3
- Only the inbound arc job, with processing time 3, needs to be scheduled, to start in [0,2]
- Total capacity into t is 5 x 2 + 2 x 1 + 4 = 16, giving an upper bound
- To achieve this, the inbound arc must carry at flow at its maximum rate for all 4 units of time that it is open. It also must carry at least 6 units of flow before closing, to “feed” the top inbound arc, so its job can’t start before time $3/2$. At this time, the storage capacity is full, so its job can’t start later, and have the arc continue to carry flow at its maximum rate.
Proposition: In the case that there is storage at some nodes, then there exists an optimal solution in which all jobs start at rational times. Further, the denominators of these optimal rational start times do not depend on parameters of the jobs (processing times, release dates or deadlines).

Proof sketch: There is a MIP model for the problem, so that once the integer variables are fixed, the constraints in the remaining LP have coefficients of \{-1, 0, 1\} or \pm u_a, where \(u_a\) is the capacity of arc \(a\). The result follows by the Fundamental Theorem of Linear Programming and Cramer’s Rule.
MIP models for dual bounds: continuous time MIP

0 ≤ t_1 ≤ t_2 ⋯ ≤ t_n = T, interval i is [t_{i-1}, t_i)

w_{ai} = 1 if job on arc a is in progress for interval i

x_{ai} = flow on arc a during interval i

x_{vi} = inventory held in storage node v at time t_i

Link time interval and processing variables:

\[ r_a w_{a(i+1)} ≤ t_i ≤ d_a + (T - d_a)(1 - w_{ai}) \]

Ensure job completed:

\[ \sum_{i=1}^{n} (t_i - t_{i-1})w_{ai} = p_a \]

Flow blocked during job:

\[ x_{ai} ≤ u_a (t_i - t_{i-1})(1 - w_{ai}) \]

variables

linearize
MIP models for dual bounds: fixed discretization

**Given** $0 \leq t_1 \leq t_2 \cdots \leq t_n = T$ an arbitrary discretization of time

$w_{ai} =$ fraction of interval $i$ for which job on arc $a$ is in progress

$y_{ai} = 1$ if job on arc $a$ starts within interval $i$

$x_{ai} =$ flow on arc $a$ during interval $i$

$x_{vi} =$ inventory held in storage node $v$ at time $t_i$

**variables**

Link time interval and processing variables:

$$(t_i - t_{i-1})w_{ai} \leq \min\{t_i, d_a\} - \max\{t_{i-1}, r_a\}$$

Ensure job completed:

$$\sum_{i=1}^{n} (t_i - t_{i-1})w_{ai} = p_a$$

Flow blocked during job:

$$x_{ai} \leq u_a(t_i - t_{i-1})(1 - w_{ai})$$
MIP models for dual bounds: fixed discretization

Given $0 \leq t_1 \leq t_2 \cdots \leq t_n = T$, interval $i$ is $[t_{i-1}, t_i)$

- $w_{ai} = \text{fraction of interval } i \text{ for which job on arc } a \text{ is in progress}$
- $y_{ai} = 1$ if job on arc $a$ starts within interval $i$
- $x_{ai} = \text{flow on arc } a \text{ during interval } i$
- $x_{vi} = \text{inventory held in storage node } v \text{ at time } t_i$

Approximating nonpreemption:

$$\sum_{k \in Q_{ai}} (t_k - t_{k-1})w_{ak} \geq p_ay_{ai}$$

and

$$\sum_{k \in P_{ai}} \mu^{-}_{ak}y_{ak} \leq (t_i - t_{i-1})w_{ai} \leq \sum_{k \in P_{ai}} \mu^{+}_{ak}y_{ak}$$

- $\mu^{-}_{13} = \frac{1}{4}(t_3 - t_2)$
- $\mu^{+}_{13} = \frac{3}{4}(t_3 - t_2)$

Min/max length of interval $i$ that job must occupy if start in interval $k$

Time intervals in which job can be in progress if start in interval $i$
Computational results on randomly generated instances

• Instances ranged from 12 nodes, 32 arcs, about 300 jobs to 64 nodes, 240 arcs and 2270 jobs
• All have $T = 1000$
• One storage node, small storage capacities are harder
• Continuous time IP struggled on truncated instances with $T = 60$ and more than 20 jobs (more than half not optimal after 2 hours CPU time)
• Bounding MIPs used either the unit discretization (1D), or only release dates and due dates (RD)
• Dual (upper) bounds: 1D < RD < 1D-LP-relaxation, but differences only slight
• Gap averages:
  ▪ < 1% on 4/8 instance classes
  ▪ < 2% on 7/8 instances classes
  ▪ < 6.7% on 8/8 instances classes
Outstanding questions

• There are many!

• Best choice of discretization for either lower or upper bounds?

• Best trade-off between granularity of discretization and rate of solution progress of MIPs?

• (Efficient) iterative discretization?

• Cutting planes to strengthen approximation to continuous time?
Outline

• Network scheduling
  ▪ Motivated by preventive maintenance scheduling in a bulk goods supply chain
  ▪ Schedule arc shutdowns in a network so as to maximize flow over time
  ▪ Storage at nodes: the impact on time
  ▪ MIP models, lower bounds and upper bounds

• Service network design
  ▪ Scheduling line-haul operations for a less-than-truckload consolidation carrier
  ▪ Fixed charge multicommodity network flow over a time-space network
  ▪ Uniform discretization approaches: the impact on optimality
  ▪ Doing better: dynamic generation of a (much smaller) time-space network
  ▪ Lower bounds, upper bounds and iterative refinement
  ▪ Computational results

• Conclusions
What does a consolidation carrier do?

• Transports shipments from origins to destinations
  ▪ Shipments are small; do not fill a whole trailer
  ▪ Shipments have a service standard

• Consolidates shipments to reduce costs
  ▪ Consolidation occurs at 2 terminal types
    – End-of-Lines (EOL - "Spoke")
    – Breakbulks (BB - "Hub")
  ▪ Cross-docking of shipments occurs at breakbulks
    – Incurs handling cost
    – Requires time: from 30 minutes to a few hours
Why does modeling time matter?

Consolidation carriers are facing customer demands for shorter and shorter service standards.

Cannot ignore time when planning freight routes and expect they’ll meet service demands.
Time-expanded networks

Time-expanded networks are frequently used to model service network design problems.

Building a time-space network requires choosing how to model time.

Fine time discretizations may lead to intractable models.

Multiple nodes for the same terminal represent dispatching at different times.

Model freight being held at a terminal or a resource idling: “holding arcs”.

“Wrap-around” arcs model that schedule will be repeated.
Fixed charge network design in the time expanded network

Given a time expanded network $$(\mathcal{N}_\mathcal{T}, \mathcal{A}_\mathcal{T})$$

\[
\text{minimize} \quad \sum_{((i,t),(j,t)) \in \mathcal{A}_\mathcal{T}} f_{ij} y_{ij}^{t\bar{t}} + \sum_{k \in \mathcal{K}} \sum_{((i,t),(j,t)) \in \mathcal{A}_\mathcal{T}} c_{ij} q_k x_{ij}^{k\bar{t}}
\]

subject to

\[
\sum_{((i,t),(j,t)) \in \mathcal{A}_\mathcal{T}} x_{ij}^{k\bar{t}} - \sum_{((j,t),(i,t)) \in \mathcal{A}_\mathcal{T}} x_{ji}^{k\bar{t}} = \begin{cases} 
1 & (i,t) = (o_k, e_k) \\
-1 & (i,t) = (d_k, l_k) \\
0 & \text{otherwise} 
\end{cases} \quad \forall k \in \mathcal{K}, (i,t) \in \mathcal{N}_\mathcal{T},
\]

\[
\sum_{k \in \mathcal{K}} q_k x_{ij}^{k\bar{t}} \leq u_{ij} y_{ij}^{t\bar{t}} \quad \forall ((i,t),(j,t)) \in \mathcal{A}_\mathcal{T},
\]

\[
x_{ij}^{k\bar{t}} \in \{0,1\} \quad \forall ((i,t),(j,t)) \in \mathcal{A}_\mathcal{T}, k \in \mathcal{K},
\]

\[
y_{ij}^{t\bar{t}} \in \mathbb{N}_\geq0 \quad \forall ((i,t),(j,t)) \in \mathcal{A}_\mathcal{T}.
\]

Based on flat network $$(N,A)$$ and time discretization $$\mathcal{T}$$

Note: integer data implies that a unit time discretization will yield an optimal continuous time solution
Why does modeling time matter?

• It takes 97 minutes to drive from Milwaukee, WI to Chicago, IL

• If time is discretized in hours (i.e. one period = one hour) how do you model this movement?

• If you model it as taking two periods, then paths that are time-feasible may be modeled as time-infeasible

• If you model it as taking one period, then consolidation opportunities may be introduced that do not really exist
Modeling time optimistically

- Experiment:
  - Solve the same SNDP using three different discretizations:
    - one period per day
    - two periods per day
    - a discretization based on past dispatches with multiple periods per day (different number of periods per day for different types of terminals: about 7 periods per day)
  - How many of the consolidation opportunities identified were “phantom”?

<table>
<thead>
<tr>
<th>Periods per day</th>
<th>Model savings (%)</th>
<th>Real savings (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>One</td>
<td>5.45</td>
<td>1.24</td>
</tr>
<tr>
<td>Two</td>
<td>4.99</td>
<td>1.73</td>
</tr>
<tr>
<td>Multiple</td>
<td>4.45</td>
<td>3.79</td>
</tr>
</tbody>
</table>
Modeling time pessimistically

• 293 instances
  ▪ Crainic et al. (2001) gives 24 flat networks
  ▪ 20-30 nodes, 230-700 arcs, 100-400 commodities
  ▪ Randomly generated service windows
  ▪ Difference schemes for consolidation flexibility
  ▪ Times given in minutes
  ▪ Horizon of about 3 days on average

• Discretizations (5, 15, 30, 60 minutes)
  ▪ Travel times round up
  ▪ Service windows round inward

• Any feasible solution found is feasible for the 1-minute problem

• What is the cost of discretization?
  ▪ “Dgap” = % cost increase due to discretization
Fundamental question

- We call the service network design problem defined over the coarsest possible discretization of time at which consolidations identified can be executed the **Continuous Time Service Network Design Problem (CTSNDP)**.

- Can an optimal “continuous” time solution be found and proved optimal without explicitly modeling every point in this discretization?
The answer is “Yes!”

• We have developed a solution approach that relies on the concept of a *partially time-expanded network* - a time expanded network in which not every time point is modeled at every node.

Network is **partially** time-expanded because location j not represented in time period 2 and 3, k not represented in period 1, i not represented in periods 1 and 3, etc.
Key mechanism

• Underestimating travel times

Actual travel time from j to k is 3

Modeled travel time can depend on the time at origin

Modeled travel time is allowed to be negative!
Theorem: If the partially time expanded network \((\mathcal{N}, \mathcal{A})\) satisfies

1. the nodes \((o_k, e_k)\) and \((d_k, l_k)\) are in \(\mathcal{N}\) for all commodities \(k\),

2. every timed copy of a flat network arc, \((i, j)\), in \(\mathcal{A}\), say \(((i, t), (j, \bar{t}))\), has \(\bar{t} \leq t + tt_{ij}\), and

3. for every arc, \((i, j)\), in the flat network and every timed copy of a node, \((i, t) \in \mathcal{T}\), there is a timed copy of \((i, j)\) in \(\mathcal{A}\) that starts at \((i, t)\),

then the fixed charge network flow problem defined on \((\mathcal{N}, \mathcal{A})\) is a relaxation of the continuous time problem, and hence its solution gives a dual (lower) bound.

Properties 2 and 3 imply that for every path in the flat network, and every timed copy of its start node, there is a path in the timed network that can be traveled in the same time or less (without the use of holding arcs).
Best dual bound for given timed nodes

Flat network

Partially time expanded network nodes

Longest arc property?

*Longest arc property:* If an arc \((i,t),(j,t')\) is in the partially time expanded network, then there does not exist a node \((j,t'')\) with \(t' < t'' \leq t + t_{ij}\)

**Theorem:** For a fixed timed node set satisfying Property 1, among the partially time expanded networks satisfying Properties 2 and 3, the one with the longest arc property induces the highest objective value.
Algorithm overview: CTSNDP-Solve

1. Find lower bound (partially time expanded network MIP)
2. Convert solution to an upper bound (LP)
3. Check if optimal
4. Add time points to lengthen at least one arc
5. Stop
Initial partially time-expanded network

<table>
<thead>
<tr>
<th>Origin</th>
<th>Destination</th>
<th>Available</th>
<th>Due</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>l</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>k</td>
<td>l</td>
<td>2</td>
<td>7</td>
</tr>
</tbody>
</table>

- Create copies of all nodes at time 1
- Create origin nodes at available time
- Create destination nodes at due time
- Add “holding” arcs
- Add travel arcs satisfying longest arc property (and Properties 2 and 3)

Nodes at time period 1 ensure that we can always add a timed copy of an arc that under-estimates travel time without having to add a new node
Validating solution

<table>
<thead>
<tr>
<th>Origin</th>
<th>Destination</th>
<th>Available</th>
<th>Due</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>l</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>k</td>
<td>l</td>
<td>2</td>
<td>7</td>
</tr>
</tbody>
</table>

This arc is “too short.” Under real travel times, consolidation on (j,3) -> (l,6) not possible.
Converting to a feasible solution

• Can the paths used in the current partially time expanded network solution be timed using true travel times, while preserving consolidations?

• Solve an LP to
  ▪ determine dispatch times for each commodity on each arc in its given path
  ▪ penalize differences in dispatch times on an arc for commodities that were consolidated on that arc in the lower bound solution

• If the LP has optimal objective value zero, STOP, this is optimal!

• Otherwise, the arcs that caused positive penalties to occur show where the travel time under-estimation is causing error

we need to lengthen at least one of these arcs
Network refinement: lengthening an arc

Choose an arc that’s too short and enables an infeasible consolidation.

Add node that enables us to model true travel time of arc and lengthen arc.

Lengthen other arcs to new node.

These operations maintain Properties 1, 2 and 3, and the longest arc property: the refined network still yields a (likely better) lower bound.
Computational tests: instances

• For each instance from before, we create multiple instances based on mapping times to a different discretization
  ▪ 60 minutes, 30 minutes, 15 minutes, 5 minutes, and 1 minute
  ▪ Express travel time in periods and round up
  ▪ Round service windows outwards
  ▪ This can create infeasible instances

• For an arc with a travel time of 140 minutes:
  ▪ 60 minute discretization => 3 periods
  ▪ 30 minute discretization => 5 periods
  ▪ 15 minute => 10 periods
  ▪ 5 minute => 18 periods
  ▪ 1 minute => 140 periods
Computational results: full discretization MIP sizes
Computational results

Relative size of partial time expanded network in final iteration

• CTSNDP-Solve maintains a much smaller network than that of the full discretization
• Increasingly so as size of problem increases
• Final network is never more than 4% of the full size for the 1-minute case

Relative size of integer program in final iteration
Computational results

- CTSNDP-Solve needs few iterations
- Half of all instances needed < 14
- Little increase with larger problems
- More than 90% of instances finished, the vast majority in under 17 minutes.
Computational results

Primal and dual gaps by iteration

Frequency of improvement
Computational results: 2-hour time, 16 GB memory limit

<table>
<thead>
<tr>
<th>Instances</th>
<th>Method</th>
<th>Time</th>
<th>Opt. Gap</th>
<th>Solved</th>
</tr>
</thead>
<tbody>
<tr>
<td>FD (\leq 16\text{GB})</td>
<td>FD</td>
<td>2,315.96</td>
<td>1.99%</td>
<td>71.49%</td>
</tr>
<tr>
<td></td>
<td>Solve-CTSNDP</td>
<td>214.85</td>
<td>0.22%</td>
<td>100.00%</td>
</tr>
<tr>
<td>FD (&gt; 16\text{GB})</td>
<td>Solve-CTSNDP</td>
<td>2,410.46</td>
<td>0.32%</td>
<td>89.62%</td>
</tr>
</tbody>
</table>

1-minute discretization
Conclusions

• For Service Network Design, there are several enhancements to consider
  ▪ Do not solve SNDP to optimality in early iterations
  ▪ Reduce iterations by making more refinements per iteration
  ▪ Parallelize iterative refinement algorithm

• Uniform discretization of time is neither necessary, nor likely to be best

• Greater accuracy in solution quality can be obtained in less computation time by handling time in new ways

• There are many interesting, outstanding questions

• To what degree to similar ideas extend to other problems?